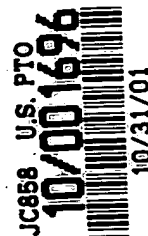```
/*   Project VendView
     SKY Wire, LP
     Copyright © 1994. All Rights Reserved.

     SUBSYSTEM: vendview.exe Application
     FILE: vndvstos.cpp
     AUTHOR: Robert M. Cowling

     OVERVIEW
     ========
     Calculate Space TO Sales

*/
#pragma hdrfile "vndvwapp.csm"
#include "vndvwhdr.h"
#pragma hdrstop

typedef
  struct
     {
        int   code;
        int   removed;
        char   product[16];
        int   velocity;
        int   capacity;
        int   optimumCapacity;
        int   newCapacity;
     } BUTTONS;

typedef
  struct
     {
        int   capacity;
        int   assigned;
        int   newAssignment;
     } COLUMNS;

// MAX_BUTTONS is defined in vndvmdi5.h as 10
#define MAX_BUTTONS12 12
//#define MAX_COLS 20

#define MAX_S2SDAYS (-90) // -90 == 90 days back
#define MIN_S2SDAYS (-14)

// Paradox engine object and database object
//
extern BEngine   *dbEngine;
extern BDatabase   *dbDatabase;

// location of common databases
extern char szCommDir[];
extern char szMapDir[];

extern char szMachStatTableName[];// = "MACHSTAT"
extern char szFacilityTableName[];// = "FACILITY"
extern char szProductTableName[];// = "PRODUCT"
extern char szMachineLoadTable[];// = "MACHLOAD"

//
//
//
long EvaluateFit(BUTTONS buttons[], int count)
     {
```

```
 67          int   index;
 68          long  answer = 0;
 69
 70          for (index = 0; index < count; index++)
 71            {
 72              long  diff = buttons[index].optimumCapacity - buttons[index].newCapacity;
 73              long  prod = diff * diff;
 74              answer += prod;
 75            }
 76          return answer;
 77        }
 78
 79
 80      #pragma argsused
 81      //  Parameters passed:
 82      //     Report code:  4 character report code - e.g. CASH - zero terminated string
 83      //     Report path:  13 to 131 character path to report file - e.g.  VVRCASH.RPT
 84      //     Print switch: True if print report
 85      //     Display switch:  True if display report
 86      //     Report title: 32 character report title - e.g. Cash Accountability
 87      //     Report parms: from 1 to 16 parameters for report
 88      //                   1.   Time of day to print report (HHMM)
 89      //                   2.   Repeat code for day to print (bits = 000000000SSFTWTM)
 90      //                   3.   From weeks (today +/- days)
 91      //                   4.   To days (today +/- days) - not used
 92      //                   5.   Amount  (two decimal positions implied)
 93      //                   6.   Number of routes (0 = none, 99 = all)
 94      //                   7.   First route
 95      //                   8.   Second route
 96      //                   9.   Third route, etc.
 97      void GenerateS2SA(char *szReportPath, BOOL bPrint, BOOL bDisplay, char *szReportTitle, int
         *nParms)
 98        {
 99
100          int   x, y, z;
101          int   columns;
102          int   buttons, oldbuttons;
103          int   venderCapacity;
104          long  venderVelocity;    // changed from int to long  RMC 2/6/96
105          int   FromDays = -(nParms[2] * 7/* Days per week */ );
106          int   minThreshold = nParms[4];
107          int   maxProducts = nParms[6];
108          int   nTmp;
109
110          CHECKHANDLES();
111
112          //
113          // GET TODAY'S DATE, INCLUDING THE DAY OF THE WEEK.
114          //
115
116          // Get the dos date.
117          struct dosdate_t today;
118          _dos_getdate(&today);
119
120          // Get it into a BDate also.
121          BDate Today;
122          BDate Search;
123          Today.year = today.year;
124          Today.month = today.month;
125          Today.day = today.day;
126          // make sure FromDays is negative, and MIN_S2SDAYS > FromDays > MAX_S2SDAYS
127          FromDays = min(MIN_S2SDAYS, FromDays);
128          FromDays = max(MAX_S2SDAYS, FromDays);
129          // calculate search date
130          IncrBDate(Today, FromDays, Search);
131
```

```
132        // Get the dos date.
133        struct dostime_t now;
134        _dos_gettime(&now);
135        if (now.hour > 7 && now.hour < 10)
136            {
137                // SGG Ask the user if they really want to run the report.
138                int RunReport = BWCCMessageBox(GetFocus(),
139                    "Space to Sales analysis will invalidate any load sheets produced earlier today.
       Do you really want to do this now?" ,
140                    "VendView Space to Sales Analysis", MB_YESNO | MB_ICONQUESTION);
141                if (RunReport != IDYES)
142                    return;
143
144                // ask for the password
145                int iRoute = 0;
146                if(VendViewAskUserPasswordDlg(GetWindowPtr(GetActiveWindow()), &iRoute).Execute()
       != IDOK)
147                    return;
148            }
149
150        // Make sure maxProducts is not ridiculously small, or greater than max number of butto
       ns.
151        if (maxProducts < 4)
152            maxProducts = 4;
153        if (maxProducts > MAX_BUTTONS12)
154            maxProducts = MAX_BUTTONS12;
155
156
157        struct
158            {
159                int nCode;
160                char szName[17];
161                int nAdds;            // additional products required
162            } stProductName[100];
163        int nProductCount;
164
165        int typeorder[3][10]    = {{0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
166                                   {0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
167                                   {0, 0, 0, 0, 0, 0, 0, 0, 0, 0}};
168        int orderindex;
169        int colagraphics;
170
171        COLUMNS col[MAX_COLS];
172
173        BUTTONS button[MAX_BUTTONS12];
174        BUTTONS oldbutton[MAX_BUTTONS12];
175
176        struct
177            {
178                int    capacity;
179                int    column;
180                int    assigned;
181            } orderedCol[MAX_COLS + MAX_BUTTONS12 + MAX_BUTTONS12 + 1];
182
183        struct
184            {
185                int    optimumCapacity;
186                int    button;
187                int    capacity;
188            } orderedButton[MAX_BUTTONS12 + 1];
189
190        char   szTable[MAXPATH];
191        BOOL   bBlank; // blank field flag
192
193        /////////////////////////////////////////////////////////////////////
194        //
```

```
195        // READ PRODUCTS INTO ARRAY
196        //
197        //
198        strcpy(szTable, szCommDir);
199        strcat(szTable, szProductTableName);
200        BCursor curProduct(dbDatabase, szTable);
201        CHECKCURSOR(&curProduct);
202        if ((curProduct.lastError == PXSUCCESS))
203           {
204           stProductName[0].nCode = 0;
205           lstrcpy(stProductName[0].szName, "NONE ASSIGNED");
206           nProductCount = 1;
207           stProductName[0].nAdds = 0;
208           curProduct.gotoBegin();
209           do
210              {
211              curProduct.gotoNext();
212              if (curProduct.lastError == PXSUCCESS)
213                 {
214                 curProduct.getRecord(); // retrieve found record
215                 ASSERT(curProduct.lastError == PXSUCCESS);
216                 BRecord *pRec = curProduct.genericRec;
217                 pRec->getField("Ident", stProductName[nProductCount].nCode, bBlank);
218                 ASSERT(pRec->lastError == PXSUCCESS);
219                 pRec->getField("Abbreviation", stProductName[nProductCount].szName, 17, bBl
ank);
220                 ASSERT(pRec->lastError == PXSUCCESS);
221                 if (bBlank)
222                    stProductName[nProductCount].szName[0] = 0;
223                 // get ranking for brand selection on maximizing products
224
225                 pRec->getField("Flavor", orderindex, bBlank);
226                 ASSERT(pRec->lastError == PXSUCCESS);
227                 if (bBlank)
228                    orderindex = 0;
229                 if ((orderindex > 0) && (orderindex < 11))
230                    typeorder[0][orderindex - 1] = stProductName[nProductCount].nCode;
231                 pRec->getField("Standard", orderindex, bBlank);
232                 ASSERT(pRec->lastError == PXSUCCESS);
233                 if (bBlank)
234                    orderindex = 0;
235                 if ((orderindex > 0) && (orderindex < 11))
236                    typeorder[1][orderindex - 1] = stProductName[nProductCount].nCode;
237                 pRec->getField("Diet", orderindex, bBlank);
238                 ASSERT(pRec->lastError == PXSUCCESS);
239                 if (bBlank)
240                    orderindex = 0;
241                 if ((orderindex > 0) && (orderindex < 11))
242                    typeorder[2][orderindex - 1] = stProductName[nProductCount].nCode;
243
244                 stProductName[nProductCount].nAdds = 0;
245                 nProductCount++;
246                 }
247              } while (curProduct.lastError == PXSUCCESS);
248           curProduct.close();
249           }
250        //
251        //
252        // end of 'READ PRODUCTS INTO ARRAY'
253        //
254        /////////////////////////////////////////////////////////////////
255
256
257        /////////////////////////////////////////////////////////////////
258        //
259        // CREATE CURSORS FOR PARADOX DATABASE ACCESS [
```

```
260        //
261        //
262
263        // Build facility table cursor.
264        strcpy(szTable, szCommDir);
265        strcat(szTable, szFacilityTableName);
266        BCursor curFacility(dbDatabase, szTable);
267        CHECKCURSOR(&curFacility);
268
269        // Build machine status table cursor.
270        strcpy(szTable, szCommDir);
271        strcat(szTable, szMachStatTableName);
272        BCursor curMachStat(dbDatabase, szTable);
273        CHECKCURSOR(&curMachStat);
274
275        // Build vender load report auxiliary data table cursor.
276        // This table contains strings needed for vender load report.
277        // Most of the data in the vender load report is taken from MACHLOAD.DB.
278        strcpy(szTable, szCommDir);
279        strcat(szTable, "VND1LOAD.DB");
280        BCursor curVenderLoad(dbDatabase, szTable);
281        CHECKCURSOR(&curVenderLoad);
282
283        // Build machine load cursor.
284        strcpy(szTable, szCommDir);
285        strcat(szTable, "MACHLOAD.DB");
286        BCursor curMachLoad(dbDatabase, szTable);
287        CHECKCURSOR(&curMachLoad);
288
289        // Build space to sales load report auxiliary data table cursor.
290        // This table contains strings needed for space to sales report.
291        // EMPTY THE TABLE.
292        EmptyTable("SP2SLOAD.DB");
293        strcpy(szTable, szCommDir);
294        strcat(szTable, "SP2SLOAD.DB");
295        BCursor curSp2SLoad(dbDatabase, szTable);
296        CHECKCURSOR(&curSp2SLoad);
297
298        // Build space to sales total load cursor.
299        // EMPTY THE TABLE.
300        EmptyTable("SP2STOTL.DB");
301        strcpy(szTable, szCommDir);
302        strcat(szTable, "SP2STOTL.DB");
303        BCursor curSp2STotal(dbDatabase, szTable);
304        CHECKCURSOR(&curSp2STotal);
305
306        //
307        //
308        // end of 'CREATE CURSORS FOR PARADOX DATABASE ACCESS' ]
309        //
310        /////////////////////////////////////////////////////////////////
311
312
313        /////////////////////////////////////////////////////////////////
314        /////////////////////////////////////////////////////////////////
315        //
316        //
317        // generate space to sales print records
318        //
319
320
321
322        if (    curFacility.lastError == PXSUCCESS
323            && curMachStat.lastError == PXSUCCESS
324            && curVenderLoad.lastError == PXSUCCESS
325            && curMachLoad.lastError == PXSUCCESS
```

```
326            && curSp2SLoad.lastError == PXSUCCESS
327            && curSp2STotal.lastError == PXSUCCESS)
328         {
329            FIELDNUMBER fld;
330            BOOL  blank;
331            BDate bdate;
332            int    nRepnum;
333            int    nextRoute;
334            int    ident;
335            int    facilIdent;
336            int    routeIdent = 0;
337
338            BRecord *machstatRec = curMachStat.genericRec;
339            BRecord *facilRec = curFacility.genericRec;
340            BRecord *vndloadRec = curVenderLoad.genericRec;
341            BRecord *machloadRec = curMachLoad.genericRec;
342            BRecord *s2sloadRec = curSp2SLoad.genericRec;
343            BRecord *totalRec = curSp2STotal.genericRec;
344
345            // fill column and button arrays
346
347            curVenderLoad.gotoBegin();
348            curVenderLoad.gotoNext();
349            while (curVenderLoad.lastError == PXSUCCESS)
350               {
351                  curVenderLoad.getRecord(vndloadRec);
352                  ASSERT(curVenderLoad.lastError == PXSUCCESS);
353
354                  // find machstat record
355                  vndloadRec->getField("Vender ident", ident, blank);
356                  ASSERT(vndloadRec->lastError == PXSUCCESS);
357                  if (blank)
358                     ident = 0;
359                  machstatRec->putField("Ident", ident);
360                  curMachStat.searchIndex(machstatRec, pxSearchFirst, 1);
361                  if (curMachStat.lastError == PXSUCCESS)
362                     {
363                        curMachStat.getRecord(machstatRec);
364
365                        // Skip if non-radio vender.
366                        machstatRec->getField("No radio", nTmp, blank);
367                        ASSERT(machstatRec->lastError == PXSUCCESS);
368                        if (blank)
369                          nTmp = 0;
370                        if (nTmp)
371                           {
372                              curVenderLoad.gotoNext();
373                              continue ;
374                           }
375
376                        // Update Velocity for Machstat; skip vender if configuration of vender has
        changed recently.
377                        if (!CalculateVelocityForVender(machstatRec, Today, Search))
378                           {
379                              curVenderLoad.gotoNext();
380                              continue ;
381                           }
382                        // find facility record
383                        machstatRec->getField("Facility Ident", facilIdent, blank);
384                        ASSERT(machstatRec->lastError == PXSUCCESS);
385                        if (blank)
386                           facilIdent = 0;
387                        facilRec->putField("Ident", facilIdent);
388                        curFacility.searchIndex(facilRec, pxSearchFirst, 1);
389                        if (curFacility.lastError == PXSUCCESS)
390                           {
```

```
391        curFacility.getRecord(facilRec);
392
393        // check for route change
394        facilRec->getField("Route Ident", nextRoute, blank);
395        if (blank)
396          nextRoute = 0;
397        if ((routeIdent) && (routeIdent != nextRoute))   // change in routes
398          {
399            // write total load records
400            // write out total additions for space to sales
401            for (x = 0; x < nProductCount; x++)
402              {
403                if (stProductName[x].nAdds > 0)
404                  {
405                    totalRec->putField("Route", routeIdent);
406                    totalRec->putField("Product code", stProductName[x].nCode);
407                    totalRec->putField("Product name", stProductName[x].szName)
408                    totalRec->putField("Count", stProductName[x].nAdds);
409                    curSp2STotal.appendRec(totalRec);
410                    // reset count
411                    stProductName[x].nAdds = 0;
412                  }
413              }
414
415          }
416        routeIdent = nextRoute;
417
418        columns = 0;
419        buttons = 0;
420        venderCapacity = 0;
421        venderVelocity = 0;
422
423        // Init col and button arrays.
424        for (x = 0; x < MAX_COLS; x++)
425          {
426            col[x].capacity = 0;
427            col[x].assigned = 0;
428            col[x].newAssignment = 0;
429          }
430        for (x = 0; x < MAX_BUTTONS12; x++)
431          {
432            button[x].code = 0;
433            button[x].removed = 0;
434            button[x].product[0] = 0;
435            button[x].capacity  = 0;
436            button[x].velocity  = 0;
437            button[x].optimumCapacity = 0;
438            button[x].newCapacity = 0;
439          }
440
441        // Get product codes.
442        fld = machstatRec->getFieldNumber("Product 1 code");
443        if (machstatRec->lastError == 0)
444          {
445            for (x = 0; x < MAX_BUTTONS12; x++)
446              {
447                machstatRec->getField(fld + x, button[x].code, blank);
448
449                if ((machstatRec->lastError) || (blank))
450                  button[x].code = 0;
451                if (button[x].code)
452                  buttons = x + 1;
453              }
454          }
455
```

```
456         // Get product names.
457         for (x = 0; x < buttons; x++)
458           {
459           for (y = 0; y < nProductCount; y++)
460             {
461             if (button[x].code == stProductName[y].nCode)
462                break;
463             }
464           if (y >= nProductCount)
465             y = 0;   // reset to no name
466           strcpy(button[x].product, stProductName[y].szName);
467           }
468
469         // Get column capacities.
470         fld = machstatRec->getFieldNumber("Column 1 capacity");
471         if (machstatRec->lastError == 0)
472           {
473           for (x = 0; x < MAX_COLS; x++)
474             {
475             machstatRec->getField(fld + x, col[x].capacity, blank);
476
477             if ((machstatRec->lastError) || (blank))
478                col[x].capacity = 0;
479             if (col[x].capacity)
480                {
481                columns = x + 1;
482                venderCapacity += col[x].capacity;
483                // fill in ordered array
484                 orderedCol[x].capacity = col[x].capacity;
485                orderedCol[x].column = x + 1;   // column no.
486                orderedCol[x].assigned = 0;
487                }
488             }
489           }
490
491         // Get column assignments and button capacities,
492         fld = machstatRec->getFieldNumber("Column 1 assigned");
493         if (machstatRec->lastError == 0)
494           {
495           for (x = 0; x < columns; x++)
496             {
497             machstatRec->getField(fld + x, col[x].assigned, blank);
498
499             if ((machstatRec->lastError) || (blank))
500                col[x].assigned = 0;
501
502             col[x].newAssignment = col[x].assigned;
503
504             // if assignment is valid, add to button capacity
505             if ((col[x].assigned > 0) && (col[x].assigned <= buttons))
506                {
507                button[col[x].assigned - 1].capacity += col[x].capacity;
508                button[col[x].assigned - 1].newCapacity += col[x].capacity;
509                // fill in ordered col array
510                orderedCol[x].assigned = col[x].assigned;
511                }
512
513             }
514           }
515         fld = machstatRec->getFieldNumber("Product 1 velocity");
516         if (machstatRec->lastError == 0)
517           {
518           for (x = 0; x < buttons; x++)
519             {
520             machstatRec->getField(fld + x, button[x].velocity, blank);
521
```

```
522              if ((machstatRec->lastError) || (blank))
523                button[x].velocity = 0;
524              venderVelocity += button[x].velocity;
525              // adjust dual assignments
526              if ((x) && (button[x].code) &&
527                  (button[x].code == button[x - 1].code) &&
528                  (button[x - 1].capacity == 0))
529                {
530                  button[x].velocity += button[x - 1].velocity;
531                  button[x - 1].velocity = 0;
532                }
533            }
534          }
535
536      // Save a copy of the original button configuration.
537
538      for (x = 0; x < buttons; x++)
539          {
540          oldbutton[x].code = button[x].code;
541          strcpy(oldbutton[x].product, button[x].product);
542          oldbutton[x].capacity  = button[x].capacity;
543          oldbutton[x].velocity  = button[x].velocity;
544          }
545      oldbuttons = buttons;
546
547      // Check for candidate vender where number of products is greater than
     min products.
548      // Get range of sales for setting maxProducts (number of recommended br
     ands) RMC 2/6/96
549      int  products = 0;
550
551      for (x = 0; x < buttons; x++)
552        if (button[x].capacity > 0)
553          products++;
554
555      // velocity total is per day (365 * 6.58 = ~100 cases per year)
556        maxProducts = 5;  // minimum maximum -- < 100     = 5
557      if (venderVelocity > 658)             // 100 - 200 = 6
558        maxProducts++;
559      if (venderVelocity > 1315)            // 200 - 300 = 7
560        maxProducts++;
561      if (venderVelocity > 1973)            // > 300      = 8
562        maxProducts++;
563
564      while (products > (maxProducts - 1))
565          {
566          // check for product velocity below threshold
567          // get minimum product with minimum velocity
568          int   minVelocity = 9999;
569          int   productWithMin;
570          for (x = 0; x < buttons; x++)
571              {
572              if (button[x].capacity > 0 && button[x].velocity <= minVelocity
     )
573                  {
574                  minVelocity = button[x].velocity;
575                  productWithMin = x;
576                  }
577              }
578
579          if (minVelocity < minThreshold)
580              {
581              venderVelocity -= button[productWithMin].velocity;
582
583              // Get count of buttons in group to be removed.
584              for (int nCountOfGroup=1; nCountOfGroup<=productWithMin; nCount
```

```
585      CfGroup++)                                 if (       button[productWithMin-nCountOfGroup].code != button[p
         roductWithMin].code
586                                                          || button[productWithMin-nCountOfGroup].capacity > 0)
587                                                    break;
588
589                                              // Give the group's columns to button 1. All columns should be
         assigned to
590                                              // the last button in the group.
591                                              button[0].newCapacity += button[productWithMin].newCapacity;
592                                              for (x=0; x<columns; x++)
593                                                 if (col[x].assigned == productWithMin + 1)
594                                                    orderedCol[x].assigned = col[x].newAssignment = col[x].assi
         gned = 1;
595
596                                              // Shift the remaining buttons below the group up.
597                                              for (x = productWithMin + 1; x < buttons; x++)
598                                                 {
599                                                 button[x-nCountOfGroup].code = button[x].code;
600                                                 strcpy(button[x-nCountOfGroup].product, button[x].product);
601                                                 button[x-nCountOfGroup].capacity = button[x].capacity;
602                                                 button[x-nCountOfGroup].velocity = button[x].velocity;
603                                                 button[x-nCountOfGroup].optimumCapacity = button[x].optimum
         Capacity;
604                                                 button[x-nCountOfGroup].newCapacity = button[x].newCapacity
         ;
605                                                 for (y=0; y<columns; y++)
606                                                    if (col[y].assigned == x + 1)
607                                                       orderedCol[y].assigned = col[y].newAssignment = col[y].
         assigned = x-nCountOfGroup+1;
608                                                 }
609
610                                              // Empty the last nCountOfGroup buttons.
611                                              for (x=buttons-nCountOfGroup;x<buttons;x++)
612                                                 {
613                                                 button[x].code = 0;
614                                                 button[x].product[0] = 0;
615                                                 button[x].capacity = 0;
616                                                 button[x].velocity = 0;
617                                                 button[x].optimumCapacity = 0;
618                                                 button[x].newCapacity = 0;
619                                                 }
620
621                                              // Reduce the number of buttons by nCountOfGroup.
622                                              buttons -= nCountOfGroup;
623                                              products--;
624                                              }
625                                           else
626                                              break;
627                                           }
628
629                                    int baseset = (maxProducts + 3) / 2;    // 4 or 5
630                                    // recommended product list  (from typeorder  -code:flag)
631                                    int productlist[10][2];
632
633                                    // check for too few products and add from standard list
634                                    if (products < maxProducts)   // need to add products
635                                       {
636                                       // what kind is this vender? flavor, standard, or diet
637                                       // get top 4 selling products from this vender
638                                       // save brand and velocity for top selling four
639                                       int  topseller[4][2] = {{0, 0}, {0, 0}, {0, 0}, {0, 0}};
640                                       for (x = 0; x < products; x++)
641                                          {
642                                          if (button[x].velocity > topseller[0][1])
643                                             {
```

```
644                         topseller[3][0] = topseller[2][0];
645                         topseller[3][1] = topseller[2][1];
646                         topseller[2][0] = topseller[1][0];
647                         topseller[2][1] = topseller[1][1];
648                         topseller[1][0] = topseller[0][0];
649                         topseller[1][1] = topseller[0][1];
650                         topseller[0][0] = button[x].code;
651                         topseller[0][1] = button[x].velocity;
652                       }
653                 else if (button[x].velocity > topseller[1][1])
654                     {
655                         topseller[3][0] = topseller[2][0];
656                         topseller[3][1] = topseller[2][1];
657                         topseller[2][0] = topseller[1][0];
658                         topseller[2][1] = topseller[1][1];
659                         topseller[1][0] = button[x].code;
660                         topseller[1][1] = button[x].velocity;
661                       }
662                 else if (button[x].velocity > topseller[2][1])
663                     {
664                         topseller[3][0] = topseller[2][0];
665                         topseller[3][1] = topseller[2][1];
666                         topseller[2][0] = button[x].code;
667                         topseller[2][1] = button[x].velocity;
668                       }
669                 else if (button[x].velocity > topseller[3][1])
670                     {
671                         topseller[3][0] = button[x].code;
672                         topseller[3][1] = button[x].velocity;
673                       }
674           }
675      //  which type matches closest  using typeorder
676      int typescore[3] = {0, 0, 0};
677      // score each type
678      for  (x = 0; x < 3; x++)
679          {
680              // score each top seller within type
681              for  (y = 0; y < 4; y++)
682                  {
683                      // score for each brand / rank for top seller within type
684                      for  (z = 0; z < 4; z++)
685                          {
686                              if (topseller[y][0] == typeorder[x][z])
687                                  typescore[x] += (4 - y) * (4 - z);
688                          }
689                  }
690          }
691          // pick winner
692      colagraphics = 0;
693      if (typescore[1] > 0)
694          {
695              colagraphics = 1;  // preset to standard
696              if ((typescore[0] > typescore[1]) && (typescore[0] > typescore
2]))
697                  colagraphics = 0;
698              if ((typescore[2] > typescore[1]) && (typescore[2] > typescore
0]))
699                  colagraphics = 2;

701              // set suggested product list; flag unusable codes
702              for (x = 0; x < 10; x++)
703                  {
704                      productlist[x][0] = typeorder[colagraphics][x];
705                      // enter initial velocity
706                      productlist[x][1] = venderVelocity / (x + 5);
707
```

```
708                            // check if used
709                            for (y = 0; y < oldbuttons; y++)
710                                {
711                                if (oldbutton[y].code == productlist[x][0])
712                                    {
713                                    // mark out all previously used products
714                                    // check if just removed
715                                    for (z = 0; z < buttons; z++)
716                                        {
717                                        if (button[z].code == productlist[x][0])
718                                            break ;
719                                        }
720                                    if (z == buttons)
721                                        productlist[x][1] = -2;   // used but dropped flag
722                                    else
723                                        productlist[x][1] = -1;   // used flag
724                                    break ;
725                                    }
726                                }
727                            }
728                        }
729                    }
730
731            // if too few products, add preferred brands
732            if (colagraphics)
733                {
734                while ((products < maxProducts) && (buttons < MAX_BUTTONS12))
735                    {
736                    int newproduct[2] = {0, 0}; // code;velocity
737                    int newproductindex = 0;
738
739                    // find suggested brand
740                    for (x = 0; x < maxProducts; x++)
741                        {
742                        // scan suggested list for available products
743                        if (productlist[x][1] > 0)
744                            {
745                            newproduct[0] = productlist[x][0];
746                            newproduct[1] = productlist[x][1];
747                            newproductindex = x;
748
749                            // break out if basic product
750                            if (x < baseset)
751                                break ;
752                            }
753                        else  // flag is -1 (used) or -2 (dropped)
754                            {
755                            // if there is a suggested product and it replaces
756                            // a dropped product, use it
757                            if ((newproduct[0]) && (productlist[x][1] == -2))
758                                break ;
759                            }
760                        }
761                    // exit sentinel -- break if no new products
762                    if (newproduct[0] == 0)
763                        break ;
764                    else
765                        {
766                        productlist[newproductindex][1] = -3;   // using suggestion
767                        }
768
769                    // fill in new product
770                    button[buttons].code = newproduct[0];
771                    // find product name
772
773                    for (x = 0; x < nProductCount; x++)
```

```
774                             {
775                                 if (stProductName[x].nCode == newproduct[0])
776                                     break;
777                             }
778                         if (x >= nProductCount)
779                             x = 0;
780                         strcpy(button[buttons].product, stProductName[x].szName);
781                         button[buttons].capacity = 1;     // mark as not dually assigned
782                         button[buttons].velocity = newproduct[1];
783                         venderVelocity += newproduct[1];
784                         button[buttons].optimumCapacity = 0;
785                         button[buttons++].newCapacity = 0;
786                         products++;
787                     }
788                 }
789
790                 // calculate optimum capacity
791                 for (x = 0; x < buttons; x++)
792                     {
793                     long opt = 0L;
794                     if (button[x].capacity > 0)   // not dual assigned )
795                         {
796                         opt = button[x].velocity;
797                         opt *= venderCapacity;
798                         if (venderVelocity > 0)
799                             opt /= venderVelocity;
800                         else
801                             opt = 1;
802                         if (opt <= 0)
803                             opt = 1L;
804                         }
805                     button[x].optimumCapacity = (int) opt;
806                     orderedButton[x].optimumCapacity = (int) opt;
807                     orderedButton[x].button = x + 1;   // button no.
808                     orderedButton[x].capacity = 0;
809                     }
810                 // order button array
811                 for (x = buttons - 1; x > 0; x--)
812                     {
813                     for (y = 0; y < x; y++)
814                         {
815                         if (orderedButton[y].optimumCapacity > orderedButton[y + 1].opt
imumCapacity)
816                             {
817                             //  swap high for low
818                             orderedButton[MAX_BUTTONS12] = orderedButton[y];
819                             orderedButton[y] = orderedButton[y + 1];
820                             orderedButton[y + 1] = orderedButton[MAX_BUTTONS12];
821                             }
822                         }
823                     }
824
825                 // add two dummy columns with zero capacity for each button
826                 for (x = 0; x < buttons; x++)
827                     {
828                     orderedCol[columns].capacity = 0;
829                     orderedCol[columns].column = 0;
830                     orderedCol[columns++].assigned = x + 1;
831                     orderedCol[columns].capacity = 0;
832                     orderedCol[columns].column = 0;
833                     orderedCol[columns++].assigned = x + 1;
834                     }
835
836                 // order column array
837                 for (x = columns - 1; x > 0; x--)
838                     {
```

```
839              for (y = 0; y < x; y++)
840                  {
841                  if (orderedCol[y].capacity > orderedCol[y + 1].capacity)
842                      {
843                      // swap high for low
844                      orderedCol[MAX_COLS + MAX_BUTTONS12 + MAX_BUTTONS12] = orde
    redCol[y];
845                      orderedCol[y] = orderedCol[y + 1];
846                      orderedCol[y + 1] = orderedCol[MAX_COLS + MAX_BUTTONS12 + M
    AX_BUTTONS12];
847                      }
848                  }
849              }
850
851          //get initial fit with old assignments
852          long fit = EvaluateFit(button, buttons);
853          long savefit = fit;
854
855          // initial new assignments
856          y = buttons - 1;
857          for (x = columns - 1; x >= 0; x--)
858              {
859              // if button is Dually Assigned with following button(s), skip it.
860              if (button[y].optimumCapacity == 0)
861                  x++;
862              else
863                  {
864                  // unassign original, reassign
865                  int asn = orderedCol[x].assigned;
866                  orderedCol[x].assigned = y + 1;   // new button assignment
867                  col[orderedCol[x].column - 1].newAssignment = y + 1;
868                  // adjust capacities
869                  button[orderedCol[x].assigned - 1].newCapacity += orderedCol[x]
    .capacity;
870                  button[asn - 1].newCapacity -= orderedCol[x].capacity;
871                  }
872              // Move back one button; wrap when needed.
873              y--;
874              if (y < 0)
875                  y = buttons - 1;
876              }
877
878          // swap routine
879          //    swap each column with all others from smallest to largest
880          savefit = EvaluateFit(button, buttons);
881          for (z = 0; z < 10; z++)
882              {
883              long keepfit = savefit;   // check in loop if optimization has been
    reached
884              for (x = columns - 1; x > 0; x--)
885                  {
886                  for (y = x - 1; y >= 0; y--)
887                      {
888                      // swap capacities if different
889                      if (orderedCol[x].capacity != orderedCol[y].capacity)
890                          {
891                          // if neither button is dually assigned
892                          if (    button[orderedCol[x].assigned - 1].optimumCapac
    ity > 0
893                              && button[orderedCol[y].assigned - 1].optimumCapac
    ity > 0)
894                              {
895                              // okay they are different, now swap and evaluate
896                              button[orderedCol[x].assigned - 1].newCapacity -= o
    rderedCol[x].capacity;
897                              button[orderedCol[x].assigned - 1].newCapacity += o
```

```
         rderedCol[y].capacity;
898                                     button[orderedCol[y].assigned - 1].newCapacity += o

         rderedCol[x].capacity;
899                                     button[orderedCol[y].assigned - 1].newCapacity -= o

         rderedCol[y].capacity;
900                                     long tryfit = EvaluateFit(button, buttons);
901
902                                     // make sure buttons have some capacity after trade
903                                     if (      tryfit < savefit
904                                         &&   button[orderedCol[x].assigned - 1].newCap
         acity > 0
905                                         &&   button[orderedCol[y].assigned - 1].newCap
         acity > 0)
906                                     {
907                                         savefit = tryfit;
908
909                                         // adjust records
910                                         int asn = orderedCol[x].assigned;
911                                         orderedCol[x].assigned = orderedCol[y].assigned
         ;
912                                         orderedCol[y].assigned = asn;
913                                         col[orderedCol[x].column - 1].newAssignment = o
         rderedCol[x].assigned;
914                                         col[orderedCol[y].column - 1].newAssignment = o
         rderedCol[y].assigned;
915                                     }
916                                     else   // set capacities back to original setting
917                                     {
918                                         button[orderedCol[x].assigned - 1].newCapacity
         += orderedCol[x].capacity;
919                                         button[orderedCol[x].assigned - 1].newCapacity
         -= orderedCol[y].capacity;
920                                         button[orderedCol[y].assigned - 1].newCapacity
         -= orderedCol[x].capacity;
921                                         button[orderedCol[y].assigned - 1].newCapacity
         += orderedCol[y].capacity;
922                                     }
923                                 }
924                             }
925                         }
926                     if (savefit == keepfit) // no change in value
927                         break;
928                 }
929         columns -= (2 * buttons);  // subtract out dummy columns count
930
931         // attempt to keep original column assignments if equal capacity
932         for (x = 0; x < columns; x++)
933             // if column x has moved to a different button
934             if (col[x].assigned != col[x].newAssignment)
935                 // Look for a column y with same capacity as column x
936                 // now assigned to column x's original button,
937                 // and not originally assigned to that same button.
938                 for (y = 0; y < columns; y++)
939                     if (      x != y
940                         &&   col[x].capacity == col[y].capacity
941                         &&   col[x].assigned == col[y].newAssignment
942                         &&   col[y].assigned != col[y].newAssignment)
943                     {
944                         // Swap columns x and y.
945                         col[y].newAssignment = col[x].newAssignment;
946                         col[x].newAssignment = col[x].assigned;
947                     }
948
949         // calculate effectiveness
950         long   origService = 10000L;
951
```

```
952        long   newService  = 10000L;
953        long   workService;
954
955        for  (x = 0; x < buttons; x++)
956           {
957           if ((button[x].velocity > 20) && (button[x].capacity > 1))
958              {
959              workService = button[x].capacity;
960              workService *= 10000;
961              workService /= button[x].velocity;
962              if (workService < origService)
963                 origService = workService;
964
965              workService = button[x].newCapacity;
966              workService *= 10000;
967              workService /= button[x].velocity;
968              if (workService < newService)
969                 newService = workService;
970              }
971           }
972
973        // Put removed products back in.
974        for  (x = 0; x < oldbuttons; x++)
975           if (button[x].code != oldbutton[x].code)
976              {
977              for (y = buttons; y > x; y--)
978                 {
979                 button[y].code = button[y-1].code;
980                 strcpy(button[y].product, button[y-1].product);
981                 button[y].velocity = button[y-1].velocity;
982                 button[y].capacity = button[y-1].capacity;
983                 button[y].optimumCapacity = button[y-1].optimumCapacity;
984                 button[y].newCapacity = button[y-1].newCapacity;
985                 }
986              buttons++;
987              button[x].code = oldbutton[x].code;
988              button[x].removed = -1;
989              strcpy(button[x].product, oldbutton[x].product);
990              button[x].velocity = oldbutton[x].velocity;
991              button[x].capacity = oldbutton[x].capacity;
992              button[x].optimumCapacity = 0;
993              button[x].newCapacity = 0;
994              }
995
996        /*
997        ASSERT(buttons == oldbuttons);
998        for (x = 0; x < oldbuttons; x++)
999           {
1000          ASSERT(button[x].code == oldbutton[x].code);
1001          ASSERT(button[x].velocity == oldbutton[x].velocity);
1002          ASSERT(button[x].capacity == oldbutton[x].capacity);
1003          }
1004       */
1005       // write out results
1006       if (((origService < 400) && (newService > (origService + 100))) ||
1007          ((origService < 700) && (newService > (origService + 300))) ||
1008          (newService > (origService + 800)))
1009          {
1010          vndloadRec->getField("Report number", nRepnum, blank);
1011          if (blank)
1012             nRepnum = 0;
1013
1014          machloadRec->putField("Report number", nRepnum);
1015          curMachLoad.searchIndex(machloadRec, pxSearchFirst, 1);
1016          if (curMachLoad.lastError == PXSUCCESS)
1017             {
```

```
1018                      curMachLoad.getRecord(machloadRec);
1019                      ASSERT(curMachLoad.lastError == PXSUCCESS);
1020
1021                      char   szWork[33];
1022
1023                      FIELDNUMBER fillEstField = machloadRec->getFieldNumber("Product
     1 estimated");
1024                      FIELDNUMBER fillS2sField = machloadRec->getFieldNumber("Product
     1 s2s");
1025                      FIELDNUMBER codeS2sField = machloadRec->getFieldNumber("Product
     1 code s2s");
1026                      FIELDNUMBER buttonS2sField = machloadRec->getFieldNumber("Butto
     n 1 assigned s2s");
1027                      FIELDNUMBER columnS2sField = machloadRec->getFieldNumber("Colum
     n 1 assigned s2s");
1028
1029                      FIELDNUMBER buttonListField  = s2sloadRec->getFieldNumber("Prod
     uct 1 button list");
1030                      FIELDNUMBER columnListField  = s2sloadRec->getFieldNumber("Prod
     uct 1 column list");
1031                      FIELDNUMBER velocityField    = s2sloadRec->getFieldNumber("Prod
     uct 1 velocity");
1032                      FIELDNUMBER capacityEstField = s2sloadRec->getFieldNumber("Prod
     uct 1 capacity est");
1033                      FIELDNUMBER capacityOptField = s2sloadRec->getFieldNumber("Prod
     uct 1 capacity opt");
1034                      FIELDNUMBER capacitys2sField = s2sloadRec->getFieldNumber("Prod
     uct 1 capacity s2s");
1035                      FIELDNUMBER productnameField = s2sloadRec->getFieldNumber("Prod
     uct 1 name");
1036
1037                      s2sloadRec->putField("Vender ident", ident);
1038                      s2sloadRec->putField("Delivery date", bdate);
1039                      s2sloadRec->putField("Report number", nRepnum);
1040
1041                      // Init all array fields in machload record to blank.
1042                      for (x = 0; x < MAX_BUTTONS; x++)
1043                        {
1044                          machloadRec->setNull(fillS2sField + x);
1045                          machloadRec->setNull(codeS2sField + x);
1046                          machloadRec->setNull(buttonS2sField + x);
1047                        }
1048                      for (x = 0; x < MAX_COLS; x++)
1049                        machloadRec->setNull(columnS2sField + x);
1050
1051                      // Init all array fields in sp2sload record to blank.
1052                      for (x = 0; x < MAX_BUTTONS; x++)
1053                        {
1054                          s2sloadRec->setNull(buttonListField + x);
1055                          s2sloadRec->setNull(productnameField + x);
1056                          s2sloadRec->setNull(columnListField + x);
1057                          s2sloadRec->setNull(velocityField + x);
1058                          s2sloadRec->setNull(capacityEstField + x);
1059                          s2sloadRec->setNull(capacityOptField + x);
1060                          s2sloadRec->setNull(capacitys2sField + x);
1061                        }
1062
1063
1064                      // Write s2s column assignments to machload record.
1065                      for (x = 0; x < MAX_COLS; x++)
1066                        machloadRec->putField(columnS2sField + x, col[x].newAssignmen
     t);
1067
1068                      // Calc delta capacities, get est fill amounts, and calc s2s fi
     ll amounts.
1069                      int nDiff[MAX_BUTTONS12];
```

```
1070                    int nUnroundedS2sFill[MAX_BUTTONS12];
1071                    int nProductCounter = 0;
1072                    for (x = 0; x < buttons; x++)
1073                      {
1074
1075                        // only for last button of each product
1076                        if (button[x].capacity > 0)
1077                          {
1078                            // Calculate delta capacity and save in delta fill arra
      y nDiff.
1079                            if (button[x].capacity == 1)
1080                              nDiff[nProductCounter] = button[x].newCapacity;
1081                            else
1082                              nDiff[nProductCounter] = button[x].newCapacity - butt
      on[x].capacity;
1083
1084                            // S2s fill = est fill + delta capacity.
1085                            machloadRec->getField(fillEstField + nProductCounter, n
      UnroundedS2sFill[nProductCounter], blank);
1086                            if (blank)
1087.                             nUnroundedS2sFill[nProductCounter] = 0;
1088                            nUnroundedS2sFill[nProductCounter] += nDiff[nProductCou
      nter];
1089                            nProductCounter++;
1090                          }
1091                      }
1092
1093                    // Round non-negative fill amounts down to nearest whole 6-pack

1094                    // Note that x here iterates over products, not buttons!
1095                    int nS2sFill[MAX_BUTTONS12];
1096                    int nTotalNonNegFill = 0;
1097                    for (x = 0; x < nProductCounter; x++)
1098                      if (nUnroundedS2sFill[x] > 0)
1099                        {
1100                          // Round down to nearest multiple of 6. Result will be >=
      0.
1101                          nS2sFill[x] = nUnroundedS2sFill[x] / 6 * 6;
1102                          // Add rounded-off fill to total of non-negative fills.
1103                          nTotalNonNegFill += nS2sFill[x];
1104                        }
1105                      else
1106                        nS2sFill[x] = nUnroundedS2sFill[x];
1107
1108                    // Decrement 2s2 non-neg fills by 6 until we have even cases on
      trip from truck to vender.
1109                    // Note that x here iterates over products, not buttons!
1110                    //
1111                    // Dependencies on variables set outside this while loop:
1112                    //   (d1) Any positive values in the first nProductCounter eleme
      nts of array nS2sFill must be multiples of 6.
1113                    //   (d2) nTotalNonNegFill must contain the sum of those positiv
      e values referred to in (1).
1114                    // Proof that while loop will halt:
1115                    //   (1) By observation of while loop conditional expression, th
      e while loop halts when
1116                    //       nTotalNonNegFill is a multiple of 24 between iterations

1117                    //   (2) We know that zero is a multiple of 6 and of 24.
1118                    //   (3) By (d2), nTotalNonNegFill is the sum of all nS2sFill >
      zero.
1119                    //   (4) By (3), nTotalNonNegFill is always >= zero.
1120                    //   (5) By (1) and (2), when nTotalNonNegFill is zero between i
      terations, while loop will halt.
1121                    //   (6) By (3), when nTotalNonNegFill > zero, there is at least
      one nS2sFill > 0.
```

```
1122                                  // (7) By (4), (5) and (6), between iterations, either the who
         le loop will halt or it will enter the
1123                                  //      next iteration guaranteed to have at least one nS2sFill
         > 0.
1124                                  // (8) By (d1), all nS2sFill > zero were rounded down to neare
         st multiples of 6.
1125                                  // (9) We know that the sum of multiples of an integer is a mu
         ltiple of that integer.
1126                                  // (a) By (3), (8) and (9), nTotalNonNegFill is always a multi
         ple of 6.
1127                                  // (b) By (6) and observation of code in body of while loop, n
         TotalNonNegFill must be decremented
1128                                  //      at least once in each iteration of the while loop.
1129                                  // (c) We know that every 4th multiple of 6 is a multiple of .
         4, so it takes at most 3 iterations
1130                                  //      to go from any multiple of 6 to a multiple of 24 by sub
         tracting 6 each iteration.
1131                                  // (d) By observation of conditional expression in for loop he
         ader,
1132                                  //      if nTotalNonNegFill reaches a multiple of 24 before the
         last iteration of the
1133                                  //      for loop, the for loop will exit without further decrem
         enting nTotalNonNegFill,
1134                                  //      allowing the while loop conditional expression to halt
         the while loop.
1135                                  // (e) By (b), (c) and (d), the while loop must halt in at most
         t 3 iterations.
1136                                  //
1137                                  while (nTotalNonNegFill % 24)
1138                                    for (x = 0; x < nProductCounter && nTotalNonNegFill % 24; x++
         )
1139                                      if (nS2sFill[x] > 0)
1140                                        {
1141                                          // Decrement both nS2sFill and nTotalNonNegFill.
1142                                          nS2sFill[x] -= 6;
1143                                          nTotalNonNegFill -= 6;
1144                                        }
1145
1146                                  // Decrease nDiffs by decreases in fills due to rounding.
1147                                  // Note that x here iterates over products, not buttons!
1148                                  for (x = 0; x < nProductCounter; x++)
1149                                    nDiff[x] -= nUnroundedS2sFill[x] - nS2sFill[x];
1150
1151                                  // Write various data for buttons and products.
1152                                  nProductCounter = 0;
1153                                  int nVacatedButtonCounter = 0; // counter for shifting product
         up into vacated buttons
1154                                  int oldMinDays = 32767;
1155                                  int newMinDays = 32767;
1156                                  for (x = 0; x < buttons; x++)
1157                                    {
1158
1159                                      // if button not being vacated
1160                                      if (!button[x].removed)
1161                                        // Write the 'product number' to which this button is 'as
         signed'.
1162                                        machloadRec->putField(buttonS2sField+x-nVacatedButtonCoun
         ter, nProductCounter+1);
1163
1164                                      // Write product values out only for last button of each pr
         oduct.
1165                                      if (button[x].capacity > 0)
1166                                        {
1167                                          // Save positive additional capacities in product array
1168                                          if (nDiff[nProductCounter] > 0)   // only for positive a
```

```
        dditicnal fill
/169
../170
/171
/172
];
/173
/174
/175
/176
/177
    S2sFill[nProductCounter]);
/178
/179
/180
    utton[x].code);
/181
/182
     newMinDays
/183
/184
/185
/186
/187
    velocity);
/188
/189
/190
/191
/192
/193
/194
    x].velocity);
/195
/196
/197
/198
/199

/200
/201
    r nos.)
/202
    ts.
/203
/204
/205
/206
/207
/208
/209
    nter);
/210
/211
/212
/213
/214
/215
/216
/217
    szWork); // (new buttons)
/218
/219
/220
    s.
/221
```

```cpp
   for (y = 0; y < nProductCount; y++)
     if (button[x].code == stProductName[y].nCode)
       {
         stProductName[y].nAdds += nDiff[nProductCounter

         break;
       }

// Write s2s fill amount to machload rec.
machloadRec->putField(fillS2sField + nProductCounter, n


// Write product code to machload rec.
machloadRec->putField(codeS2sField + nProductCounter, b


// Get old days and new days, and update oldMinDays and

int days;

if (button[x].capacity > 1)
   {
     days = MulDiv(button[x].capacity, 10000, button[x].

     if (days < oldMinDays)
       oldMinDays = days;
   }
// Only update newMinDays if product not being removed.
if (!button[x].removed)
   {
     days = MulDiv(button[x].newCapacity, 10000, button[

     if (days < newMinDays)
       newMinDays = days;
   }

// Write button string  (eg. "2, 3, 4") to sp2sload rec

// For removed low-vel products, write "Remove".
// Shift button numbers for remaining products up (lowe
// to what they will be after removal of low-vel produc

if (button[x].removed)
   strcpy(szWork, "Remove");
else
   for (y = 0; y <= x; y++)
     {
       char  szBut[5];
       wsprintf(szBut, ", %d", y + 1 - nVacatedButtonCou

       if ((y) &&
           (button[y].code == button[y - 1].code) &&
           (button[y - 1].capacity == 0))
         strcat(szWork, szBut);
       else
         strcpy(szWork, &szBut[2]);
     }
s2sloadRec->putField(buttonListField + nProductCounter,


// Write vel, cap, opt cap and new cap to sp2sload rec.
// Opt cap and new cap will be zero for removed product

s2sloadRec->putField(velocityField + nProductCounter, b
```

```
1222        uttcn[x].velocity);               int buttoncapacity = (button[x].capacity == 1) ? 0 : bu

1223      tton[x].capacity;                   s2sloadRec->putField(capacityEstField + nProductCounter

1224    , buttoncapacity);                    s2sloadRec->putField(capacityOptField + nProductCounter

1225    , button[x].optimumCapacity);         s2sloadRec->putField(capacitys2sField + nProductCounter

1226    , button[x].newCapacity);             s2sloadRec->putField(productnameField + nProductCounter

1227    , button[x].product);
1228                                          // write product's column list to sp2sload rec.
1229                                          if (button[x].removed)
1230    );                                      s2sloadRec->setNull(columnListField + nProductCounter

1231                                          else
1232                                            {
1233                                              szWork[0] = 0;
1234                                              if (button[x].newCapacity)
1235                                                for (y = 0; y < columns; y++)
1236                                                  {
1237                                                    if (col[y].newAssignment == x + 1 - nVacated8
        uttonCounter)
1238                                                      {
1239                                                        char   szCol[5];
1240
1241                                                        wsprintf(szCol, ", %d", y + 1);
1242                                                        if (szWork[0])
1243                                                          strcat(szWork, szCol);
1244                                                        else
1245                                                          strcpy(szWork, &szCol[2]);
1246                                                      }
1247                                                  }
1248                                              s2sloadRec->putField(columnListField + nProductCoun
        ter, szWork);
1249                                            }
1250
1251                                            nProductCounter++;
1252                                          } // if was last button of a product
1253
1254                                          // If button is being vacated, update vacated buttons count
        er.
1255                                          if (button[x].removed)
1256                                            nVacatedButtonCounter++;
1257
1258                                        } // for each button
1259
1260                                        s2sloadRec->putField("Old days left", oldMinDays);
1261                                        ASSERT(s2sloadRec->lastError == PXSUCCESS);
1262                                        s2sloadRec->putField("New days left", newMinDays);
1263                                        ASSERT(s2sloadRec->lastError == PXSUCCESS);
1264
1265                                        nTmp = 1; // init space to sales flag to true
1266                                        machloadRec->putField("S2S", nTmp);
1267                                        ASSERT(PXSUCCESS == machloadRec->lastError);
1268
1269                                        // Update machload rec.
1270                                        curMachLoad.updateRec(machloadRec);
1271                                        // Append Sp2SLoad rec to table.
1272                                        curSp2SLoad.appendRec(s2sloadRec);
1273                                      }
1274
1275                        } // if min days increased enough
1276                    } // if curFacility.lastError == PXSUCCESS
1277                } // if curMachStat.lastError == PXSUCCESS
```

```
1278                  curVenderLoad.gotoNext();
1279
1280              } // while curVenderLoad.lastError == PXSUCCESS
1281
1282          // write out total additions for sp 2 sales
1283          for (x = 0; x < nProductCount; x++)
1284              {
1285              if (stProductName[x].nAdds > 0)
1286                  {
1287                  totalRec->putField("Route", routeIdent);
1288                  totalRec->putField("Product code", stProductName[x].nCode);
1289                  totalRec->putField("Product name", stProductName[x].szName);
1290                  totalRec->putField("Count", stProductName[x].nAdds);
1291                  curSp2STotal.appendRec(totalRec);
1292                  }
1293              } // for
1294          } // if all cursors created successfully
1295
1296      curFacility.close();
1297      curMachStat.close();
1298      curVenderLoad.close();
1299      curMachLoad.close();
1300      curSp2SLoad.close();
1301      curSp2STotal.close();
1302
1303
1304      CHECKHANDLES();
1305      /////////////////////////////////////////////////////////////////////
1306  } // GenerateS2SA()
```